Processes in Distributed Systems

Distributed Systems
Sistemi Distribuiti

Andrea Omicini
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2012/2013

Outline

These Slides Contain Material from [Tanenbaum and van Steen, 2007]

Slides were made kindly available by the authors of the book

- Such slides shortly introduced the topics developed in the book [Tanenbaum and van Steen, 2007] adopted here as the main book of the course
- Most of the material from those slides has been re-used in the following, and integrated with new material according to the personal view of the teacher of this course
- Every problem or mistake contained in these slides, however, should be attributed to the sole responsibility of the teacher of this course

What You Are Supposed to Know from the Operating Systems Course

Basics about processes

- Processes
- Threads
- Light-weight processes

Basics about virtualisation

- Concept of virtualisation
- Basic architectures of virtual machines

What You Are Supposed to Know from the Operating Systems Course

Basics about processes

- Processes
- Threads
- Light-weight processes

Basics about virtualisation

- Concept of virtualisation
- Basic architectures of virtual machines

Outline

Threads in Non-distributed Systems

Benefits

- Non-blocking behaviour

→ A blocking system call can be implemented with no need of blocking an entire application

- Exploiting parallelism

→ A multi-processor unit can be exploited at its best

- Exploiting shared data space communication in large applications

→ A multi-threaded application is more efficient in switching than a multi-process one

- Software engineering expressive power

→ A multi-threaded application models a concurrent cooperative application straightforwardly

Threads in Distributed Systems

Main issue

- Processes in distributed systems are really concurrent—no processor virtualisation needed
- However, coupling in distributed systems is even more dangerous than in traditional ones

$\leftarrow$ Possible chain effect throughout the network, thus hindering the main benefit of distribution

- Non-blocking behaviour is then the most relevant benefit here

$\rightarrow$ A blocking call would not block an entire distributed application

Examples

- Multi-threaded clients
- Multi-threaded servers

Threads in Distributed Systems

Main issue

- Processes in distributed systems are really concurrent—no processor virtualisation needed
- However, coupling in distributed systems is even more dangerous than in traditional ones

$\leftarrow$ Possible chain effect throughout the network, thus hindering the main benefit of distribution

- Non-blocking behaviour is then the most relevant benefit here

$\rightarrow$ A blocking call would not block an entire distributed application

Examples

- Multi-threaded clients
- Multi-threaded servers

Multi-threaded Clients

Issue and basic idea

- In a WAN, latencies could be heavy

$\leftarrow$ Distribution transparency is definitely an issue

- Typical approach: start communication and immediately proceed doing something else

$\rightarrow$ No apparent wait

Example of a Multi-threaded Client

Web browser

- Many TCP/IP connection for a single HTML page
- Each one served through its own connection, handled by a separate thread
- Also, visualisation of different element can be threaded

Benefits

- Perceived response is almost immediate
- Simple architecture

← Each connection thread is actually very simple

- Exploitation of replicated servers

← Each connection could be potentially served by the a different server, according to availability

Example of a Multi-threaded Client

Web browser

- Many TCP/IP connection for a single HTML page
- Each one served through its own connection, handled by a separate thread
- Also, visualisation of different element can be threaded

Benefits

- Perceived response is almost immediate
- Simple architecture

← Each connection thread is actually very simple

- Exploitation of replicated servers

← Each connection could be potentially served by the a different server, according to availability

Multi-threaded Servers

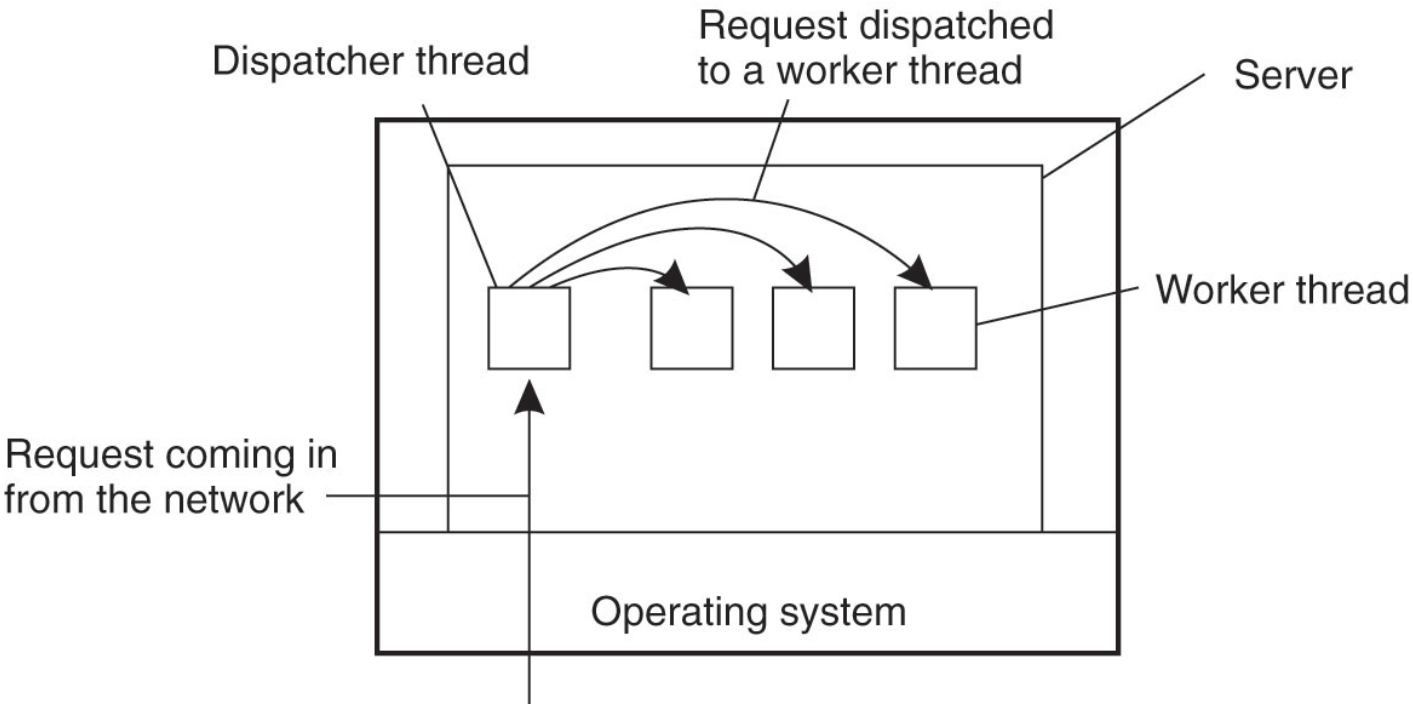
Benefits

- More benefits for servers than for clients
- Multithreading simplifies server code

← When multiple services should be granted to multiple clients

- Better performance
- Better exploitation of multi-processor architectures

An Architecture for a Multi-threaded Server



A multithreaded server organised in a dispatcher/worker model [Tanenbaum and van Steen, 2007]

Outline

The Main Idea

Resource virtualisation

- Not only many processes on one actual processor—shown as a pool of virtual processors
- Any resource could be shared and viewed like that
- In distributed systems this is of particular interest, given the need for transparency

Virtualisation in Distributed Systems

Issues

Porting of legacy Legacy software comes form the relatively low change rate of high-level software, while hardware and low-level systems change quite fast—to keep high-level software working, virtualisation is of help

Porting through network Heterogeneous computing platforms are interconnected and should make diverse applications run—which could bring their own environment with them

Replication A server could be completely replicated whenever and wherever needed—e.g., edge servers

Outline

Moving Code

Sometimes passing data is not enough

- Sometimes we would like to change the place where the code is executed—for load balancing, security, scalability, ...
- Sometimes we do not like to separate the data from the code to be executed on them (e.g., objects, agents)

→ Then, passing data between processes is no longer enough

→ Code should be passed

Reasons for Migrating Code

Process migration

- Traditionally, code is moved along with the whole computational context
- Moving code is typically moving processes [Milojicic et al., 2000]

Why?

- Load balancing
- Minimising communication
- Optimising perceived performance
- Improving scalability
- Flexibility through dynamic configurability
- Improving fault tolerance

Reasons for Migrating Code

Process migration

- Traditionally, code is moved along with the whole computational context
- Moving code is typically moving processes [Milojicic et al., 2000]

Why?

- Load balancing
- Minimising communication
- Optimising perceived performance
- Improving scalability
- Flexibility through dynamic configurability
- Improving fault tolerance

Models for Code Migration

There is much more than just moving code

- What do we move along with a program?
- Execution status, pending signals, data, ...

Understanding code mobility [Fuggetta et al., 1998]

- A process can be thought as three segments

  code segment the set of the executable instructions of the process

  resource segment the set of the references to the external resources needed by the process—like files, printers, devices, other processes, ...

  execution segment the store for the execution state of the process—with private data, stack, program counter

- Depending on what is moved along with the code, we can classify different types of code mobility

Models for Code Migration

There is much more than just moving code

- What do we move along with a program?
- Execution status, pending signals, data, ...

Understanding code mobility [Fuggetta et al., 1998]

- A process can be thought as three segments

  code segment the set of the executable instructions of the process

  resource segment the set of the references to the external resources needed by the process—like files, printers, devices, other processes, ...

  execution segment the store for the execution state of the process—with private data, stack, program counter

- Depending on what is moved along with the code, we can classify different types of code mobility

Weak Mobility

The bare minimum for code migration

- Only the code segment is transferred
- Possibly along with some initialisation data

Main idea

- The code can be executed every time *ex novo*
- So, we do not care about any computational context
- Or, maybe, the computational context we need is the target one

Main benefit

- The only requirement is that the target machine can execute the code
- Weak mobility is very simple
- It has no particular restrictions or further requirements to be implemented

Weak Mobility

The bare minimum for code migration

- Only the code segment is transferred
- Possibly along with some initialisation data

Main idea

- The code can be executed every time *ex novo*
- So, we do not care about any computational context
- Or, maybe, the computational context we need is the target one

Main benefit

- The only requirement is that the target machine can execute the code
- Weak mobility is very simple
- It has no particular restrictions or further requirements to be implemented

Weak Mobility

The bare minimum for code migration

- Only the code segment is transferred
- Possibly along with some initialisation data

Main idea

- The code can be executed every time *ex novo*
- So, we do not care about any computational context
- Or, maybe, the computational context we need is the target one

Main benefit

- The only requirement is that the target machine can execute the code
- Weak mobility is very simple
- It has no particular restrictions or further requirements to be implemented

Strong Mobility

Moving execution context

- The execution segment is transferred along with the code segment

Main benefit

- A process can be stopped, moved, and then restart on another machine

Requirements

- Strong mobility is very demanding
- Technological environment should allow for it

Strong Mobility

Moving execution context

- The execution segment is transferred along with the code segment

Main benefit

- A process can be stopped, moved, and then restart on another machine

Requirements

- Strong mobility is very demanding
- Technological environment should allow for it

Strong Mobility

Moving execution context

- The execution segment is transferred along with the code segment

Main benefit

- A process can be stopped, moved, and then restart on another machine

Requirements

- Strong mobility is very demanding
- Technological environment should allow for it

Sender- vs. Receiver- Initiated Migration

Sender-initiated migration

- Migration is initiated where the code resides / is being currently executed
- Examples: search-bots, mobile agents
- Servers should know clients, and ensure security of resources

→ More complex interaction scheme

Client-initiated migration

- Migration is initiated by the target machine, requiring a new behaviour to be added
- Examples: Java Applets, JavaScript chunks
- Just a few resources on clients need to be secured
- Clients may also be anonymous

→ Less complex interaction scheme

Sender- vs. Receiver- Initiated Migration

Sender-initiated migration

- Migration is initiated where the code resides / is being currently executed
- Examples: search-bots, mobile agents
- Servers should know clients, and ensure security of resources

→ More complex interaction scheme

Client-initiated migration

- Migration is initiated by the target machine, requiring a new behaviour to be added
- Examples: Java Applets, JavaScript chunks
- Just a few resources on clients need to be secured
- Clients may also be anonymous

→ Less complex interaction scheme

Separate vs. Target Process Execution

Weak mobility and execution of mobile code

- In the case of weak mobility, one may execute the mobile code on either the target process or a separate process
- For instance, Java Applets are executed in the browser's address space

→ No need for inter-process communication at the target machine

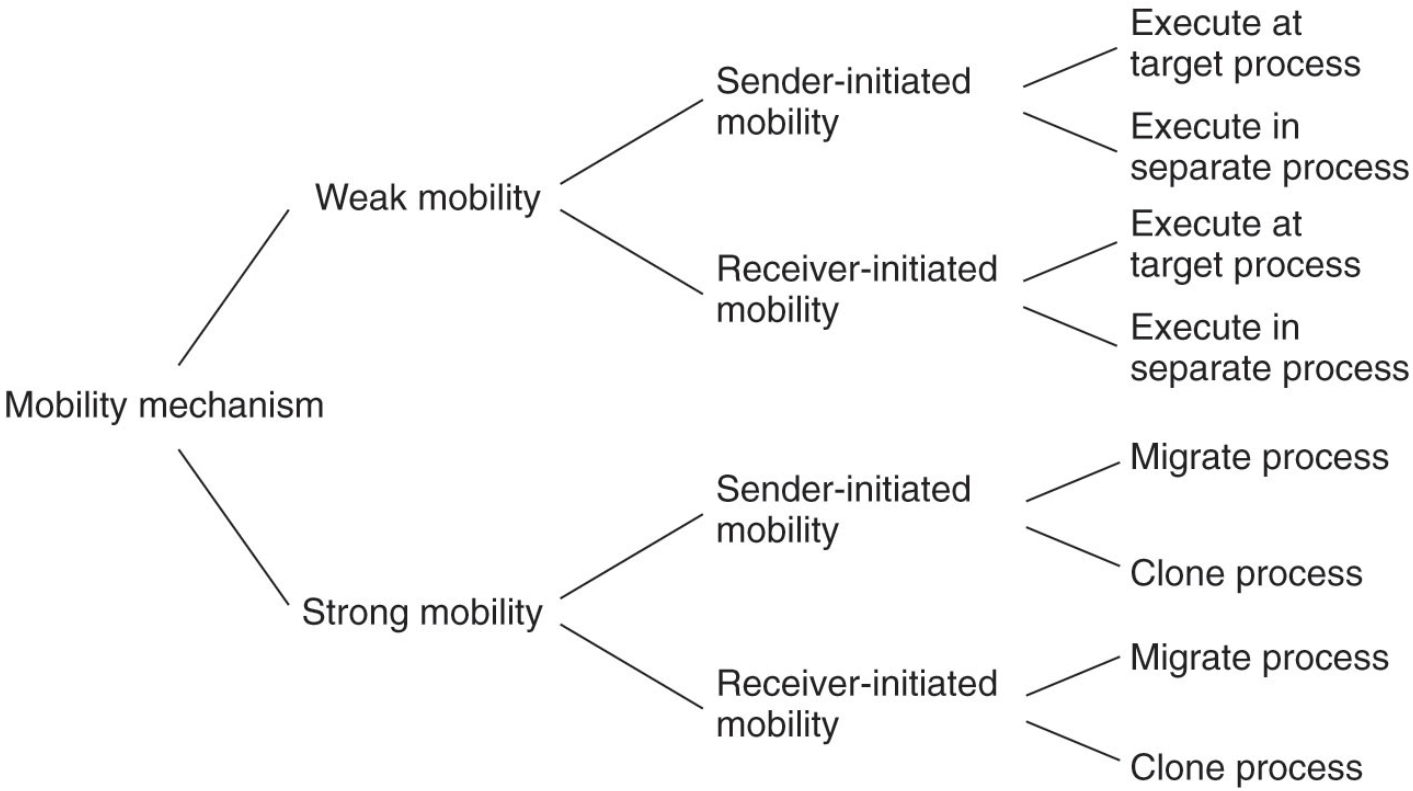
- Main problem: protection against malicious or buggy code execution
- Solution: assigning mobile code execution to a separated process

Cloning vs. Migrating

Strong mobility can be supported also by remote cloning

- Cloning yields an exact copy of the original process, executed on the target machine
- Cloned process is executed in parallel to the original process, on different machines
- Example: In UNIX, forking a child process and let it execute on a remote machine
- Cloning is an alternative to migration
- Cloning in some sense improve distribution transparency, in that the processes are transparently replicated on many different machines

Models for Code Migration



Alternatives for code migration
[Tanenbaum and van Steen, 2007]

Migration and Local Resources

Migration and the resource segment

- Till now, we have only accounted for migration of the code and execution segments
- Main problem: resources might not be as easy to move around as code and variables
- Example: A huge database might in theory be moved across the network, but in practice it will not
- Either references need to be updated, or resources need to be moved

Two issues

- How does the resource segment refer to resources?
- How does the resource relate with the hosting machine?

Migration and Local Resources

Migration and the resource segment

- Till now, we have only accounted for migration of the code and execution segments
- Main problem: resources might not be as easy to move around as code and variables
- Example: A huge database might in theory be moved across the network, but in practice it will not
- Either references need to be updated, or resources need to be moved

Two issues

- How does the resource segment refer to resources?
- How does the resource relate with the hosting machine?

How does the resource segment refer to resources?

Process-to-resource binding

Binding by identifier Need of a resource with a given name – e.g., via an URL, or a local ID

Binding by value Need of a resource based on its value – e.g., code libraries

Binding by type Need of a resource based on its type – typically, local devices like printers, monitors, ...

How does the resource relate with the hosting machine?

Resource-to-machine binding

Unattached resources Resources that can be easily moved between different machines – like, files associated to the migrating code

Fastened resources Resources that can be moved, but at a cost – like, a local database

Fixed resources Resources bounded to a specific machine – like, a monitor

Code Migration and Local Resources

		Resource-to-machine binding		
		Unattached	Fastened	Fixed
Process-to-resource binding	By identifier	MV (or GR)	GR (or MV)	GR
	By value	CP (or MV,GR)	GR (or CP)	GR
	By type	RB (or MV,CP)	RB (or GR,CP)	RB (or GR)

GR	Establish a global systemwide reference
MV	Move the resource
CP	Copy the value of the resource
RB	Rebind process to locally-available resource

Actions to be taken with respect to the references to local resources when migrating code to another machine
[Tanenbaum and van Steen, 2007]

Summing Up

Processes in distributed systems

- Processes and threads retain and further develop their importance in distributed systems
- Virtualisation gain even more importance in the distributed setting
- Whatever we have learned on these issues still holds, but an original look should be given to understand their relevance in distributed systems

Code migration

- Code may move through distributed machines for a number of good reasons
- Different types of code mobility are possible, depending on either the application needs or the technology constraints

Summing Up

Processes in distributed systems

- Processes and threads retain and further develop their importance in distributed systems
- Virtualisation gain even more importance in the distributed setting
- Whatever we have learned on these issues still holds, but an original look should be given to understand their relevance in distributed systems

Code migration

- Code may move through distributed machines for a number of good reasons
- Different types of code mobility are possible, depending on either the application needs or the technology constraints

References I

Fuggetta, A., Picco, G. P., and Vigna, G. (1998).
Understanding code mobility.
IEEE Transactions on Software Engineering, 24(5):342–361.

Milojicic, D. S., Douglis, F., Paindaveine, Y., Wheeler, R., and Zhou, S. (2000).
Process migration.
ACM Computing Surveys, 32(3):241–299.

Tanenbaum, A. S. and van Steen, M. (2007).
Distributed Systems. Principles and Paradigms.
Pearson Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition.

Processes in Distributed Systems

Distributed Systems
Sistemi Distribuiti

Andrea Omicini
andrea.omicini@unibo.it

Dipartimento di Informatica – Scienza e Ingegneria (DISI)
Alma Mater Studiorum – Università di Bologna a Cesena

Academic Year 2012/2013